



DIGITAL
INNOVATION
ONE

Orientação a Objetos

Thiago Leite e Carvalho

Engenheiro de Software, Professor, Escritor

Mais sobre mim

- Java, backend, docência, livros, artigos e cursos
- Mestre e Serpro
- O que me motiva?
- Pizzas e massas, cerveja e futebol



Mais sobre mim

- <https://www.linkedin.com/in/thiago-leite-e-carvalho-1b337b127/>
- <https://github.com/thiagoleitecarvalho>
- <https://github.com/tlcdio>

Objetivo do curso

Possibilitar que o aluno compreenda todos os conceitos relativos à Orientação a Objetos(OO).

Percurso

Aula 1

Porque usar?

Aula 2

Os fundamentos

Aula 3

A estrutura

Aula 4

As relações

Percurso

Aula 5

A Organização

Aula 6

Próximos passos

Requisitos

- ✓ Lógica de Programação
- ✓ Vontade de aprender

Dúvidas durante o curso?

- > Fórum do curso
- > Comunidade online (discord)

Aula 1: Porque usar?

Orientação a
Objetos

Objetivos

1. Explicar porque devemos programar orientado a objetos?

PE vs POO

- Paradigma Estruturado tem uma representação mais simplista
- Paradigma Orientado a Objeto tem uma representação mais realista

PE vs POO

- Paradigma Estruturado foca em operações(funções) e dados
- Paradigma Orientado a Objetos foca na modelagem de entidades e nas interações entre estas

PE vs POO

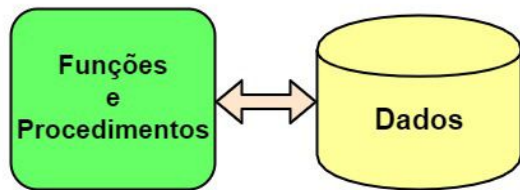
- Programação Estruturada foca mais no "como fazer"
- Programação Orientada a Objetos foca mais no "o que fazer"

Vantagens da POO

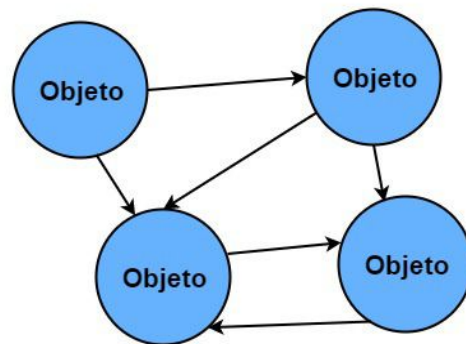
- Melhor coesão
- Melhor acoplamento
- Diminuição do Gap semântico
- Coletor de lixo



Resumo



Programação Estruturada



Programação Orientada a Objetos

Aula 2: Os fundamentos

Orientação a
Objetos

Objetivos

1. Definir o que é a OO?
2. Entender os pilares no qual o paradigma se sustenta.

Definição

"A Orientação a Objetos é um paradigma de análise, projeto e programação de sistemas de software baseado na composição e interação entre diversas unidades de software chamadas de objetos."

(https://pt.wikipedia.org/wiki/Orientação_a_objetos)

Fundamentos

- Abstração

"Processo pelo qual se isolam características de um objeto, considerando os que tenham em comum certos grupos de objetos."

Fundamentos

- Reuso

"Capacidade de criar novas unidades de código a partir de outras já existentes."

Fundamentos

- Encapsulamento

"Capacidade de esconder complexidades e proteger dados."

Exercitando

Levando em consideração uma loja on-line de livros, modele uma entidade livro.



Exercitando

Livro

Quantidade
de páginas

Ano
publicação

Peso

Edição

Tema

Assunto

Editora

Tipo de
Papel

Tipo de
Capa

ISBN

Título

Autor

Idioma

Gramatúra

Sub-Título

Dimensões

Quantidade
de
capítulos

Acabamento

Coloração



Exercitando

Livro

Tema

Autor

Editora

Sub-Título

Edição

ISBN

Ano
publicação

Quantidade
de páginas

Assunto

Título



Exercitando

Livro

Quantidade
de páginas

Tipo de
Papel

Peso

Tipo de
Capa

Coloração

Acabamento

ISBN

Gramatúra

Sub-Título

Dimensões



Exercitando

Livro

Quantidade
de páginas

Assunto

ISBN

Título

Editora

Autor

Aula 3: A estrutura

Orientação a
Objetos

Objetivos

1. Apresentar os conceitos que criam as estruturas básicas da OO:
 - Classe
 - Atributo
 - Método
 - Objeto
 - Mensagem

Classe

"É uma estrutura que abstrai um conjunto de objetos com características similares. Uma classe define o comportamento de seus objetos através de métodos e os estados possíveis destes objetos através de atributos. Em outros termos, uma classe descreve os serviços providos por seus objetos e quais informações eles podem armazenar."

([https://pt.wikipedia.org/wiki/Classe_\(programação\)](https://pt.wikipedia.org/wiki/Classe_(programação)))

Classe

- Exemplos:

Bola

Carro

Viagem

Computador

Venda

Comprador

- Dicas:
 - Substantivos
 - Nome significativos
 - Contexto deve ser considerado

Classe

- Códigos:

Java

```
class Carro {  
  
}
```

C#

```
class Carro {  
  
}
```

Python

```
class Carro:  
    pass
```


Exercitando 1

Crie a classe "Carro".

Obs: use a linguagem que gostar e siga as dicas sobre como criar classes

Atributo

"É o elemento de uma classe responsável por definir sua estrutura de dados. O conjunto destes será responsável por representar suas características e fará parte dos objetos criados a partir da classe."



Atributo

- Exemplos:

Bola

- diametro

Carro

- cor

Viagem

- distancia

Computador

- memoria

Venda

- valor

Comprador

- nome

Atributo

Atributo x Variável

A: O que é próprio e peculiar a alguém ou a alguma coisa.

V: Sujeito a variações ou mudanças; que pode variar;
inconstante, instável

Atributo

- Dicas:
 - Substantivos e adjetivos
 - Nome significativos
 - Contexto deve ser considerado
 - Abstração
 - Tipos adequados

Atributo

- Códigos:

Java

```
class Carro {  
    int portas;  
}
```

C#

```
class Carro {  
    int portas;  
}
```

Python

```
class Carro:  
    portas = 0
```

Exercitando 2

Evolua o exercicio 1 e defina 3 atributos para sua classe "Carro": cor, modelo e capacidade do tanque.

Obs: use a linguagem que gostar e siga as dicas sobre como criar atributos



Método

"É uma porção de código (sub-rotina) que é disponibilizada pela classe. Este é executado quando é feita uma requisição a ele. Um método serve para identificar quais serviços, ações, que a classe oferece. Eles são responsáveis por definir e realizar um determinado comportamento"

Método

- Exemplos:

Carro

- ligar

Computador

- desligar

Venda

- calcular total

Comprador

- realizar troca

Método

- Criação:

Java e C#

- Visibilidade
- Retorno
- Nome
- Parâmetros

Python

- `def`
- Nome
- Parâmetros

Método

- Dicas:
 - Verbos
 - Nome significativos
 - Contexto deve ser considerado

Método

- Códigos:

Java

```
class Carro {  
    void frear() {  
        ...  
    }  
}
```

C#

```
class Carro {  
    void frear() {  
        ...  
    }  
}
```

Python

```
class Carro:  
    def frear()  
        ...
```

Método

- Dois métodos especiais:
 - Construtor
 - Destruitor

Método

- Dois métodos especiais:



Construtor

Java

C#

Python

```
class Carro {      class Carro {      class Carro:
    Carro() {      Carro() {      def __init__(self):
        ...
    }
}
```



Método

- Dois métodos especiais:



Destruitor
Java

C#

Python

```
class Carro {  
    void finalize() {  
        ...  
    }  
}
```

```
class Carro {  
    ~Carro() {  
        ...  
    }  
}
```

```
class Carro:  
    def __del__(self):  
        ...
```

Método

- Sobrecarga:
 - Mudar a assinatura de acordo com a necessidade

- Assinatura: nome + parâmetros

Java

```
m1()
```

```
m1(int i)
```

```
m1(float f)
```

```
m1(String s, long l)
```

```
m1(long l, String s)
```

C#

```
M1()
```

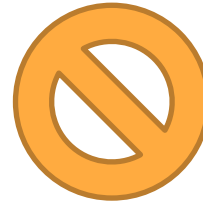
```
M1(int i)
```

```
M1(float f)
```

```
M1(String s, long l)
```

```
M1(long l, String s)
```

Python



- Porque usar?

Exercitando 3

Evolua o conceito do exercício 2 e defina 1 método para calcular o valor total para encher o tanque. Este deve receber como parâmetro o valor da gasolina. Faça também duas sobrecargas do construtor.

Obs: use a linguagem que gostar e siga as dicas sobre como criar métodos

Obs: Crie métodos específicos para fornecer e obter os valores dos atributos(set/get), caso aplicável.

Objeto

"Um objeto é a representação de um conceito/entidade do mundo real, que pode ser física (bola, carro, árvore etc.) ou conceitual (viagem, estoque, compra etc.) e possui um significado bem definido para um determinado software. Para esse conceito/entidade, deve ser definida inicialmente uma classe a partir da qual posteriormente serão instanciados objetos distintos."

Objeto

- Criação:

Java

```
Carro carro = new Carro();
```

Python

```
carro = Carro()
```

C#

```
Carro carro = new Carro();
```

Mensagem

"É o processo de ativação de um método de um objeto. Isto ocorre quando uma requisição (chamada) a esse método é realizada, assim disparando a execução de seu comportamento descrito por sua classe. Pode também ser direcionada diretamente à classe, caso a requisição seja a um método estático."

Mensagem

Java

```
Carro carro = new Carro();  
carro.<método>;
```

```
Carro.<método>;
```

C#

```
Carro carro = new Carro();  
carro.<método>;
```

```
Carro.<método>;
```

Python

```
carro = Carro()  
carro.<método>
```

```
Carro.<método>
```



Siga em frente...

- Instância x Estático: atributos e métodos
- Estado de um Objeto
- Identidade de um Objeto
- Representação numérica de um objeto
- Representação padrão de um objeto

Exercitando 4

Evolua o conceito do exercício 3 criando objetos da classe "Carro". Use os métodos set/get, quando aplicáveis, para definir os valores dos atributos e exibir estes valores "get". Passe também uma mensagem para o cálculo do total pra encher o tanque.

Obs: use a linguagem que gostar e siga as dicas sobre como criar métodos, atributos, etc.

Obs: use `System.out`, `Console.WriteLine` ou `print`

Aula 4: As relações

Orientação a
Objetos

Objetivos

1. Apresentar os conceitos que ajudam a criar entidades a partir de outras entidades:

- Herança
- Associação
- Interface

Herança

"É o relacionamento entre classes em que uma classe chamada de subclasse (classe filha, classe derivada) é uma extensão, um subtipo, de outra classe chamada de superclasse (classe pai, classe mãe, classe base). Devido a isto, a subclasse consegue reaproveitar os atributos e métodos dela. Além dos que venham a ser herdados, a subclasse pode definir seus próprios membros."

Herança

- Códigos:

Java

```
class A extends B {  
    ...  
}
```

C#

```
class A : B {  
    ...  
}
```

Python

```
class A(B):  
    ...
```

Exercitando 1

Crie a classe "Veiculo", "Carro", "Moto" e "Caminhao".

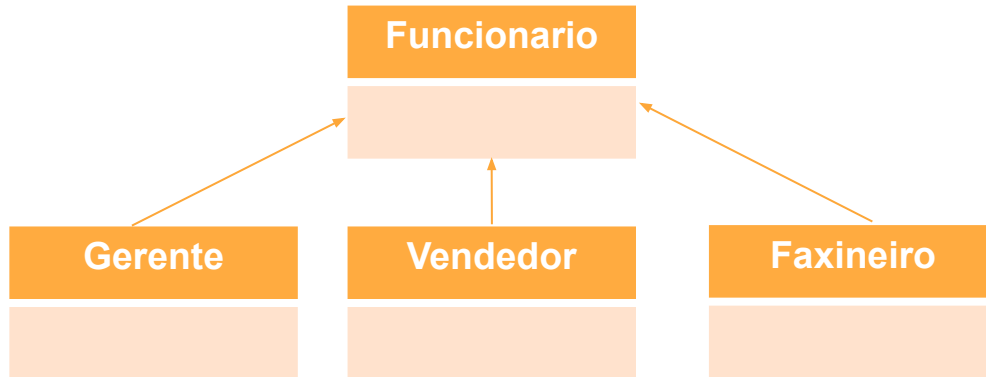
Obs: use a linguagem que gostar e siga as dicas sobre como criar classes. Faça a relação de herança que julgue adequada.

Herança

- Tipos de herança:
 - Simples
 - Múltipla

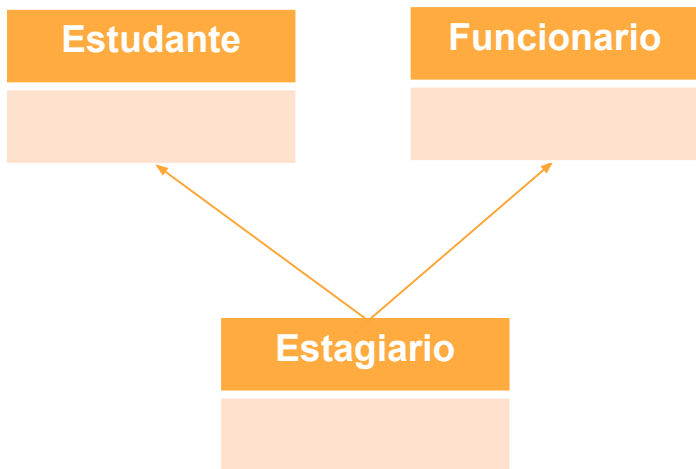
Herança

- Tipos de herança:
 - Simples
 - A classe filha tem só uma classe mãe



Herança

- Tipos de herança:
 - Múltipla
 - A classe filha tem uma ou mais classes mães



Herança

- Tipos de herança:

- Múltipla

- Java 

- C# 

- Python

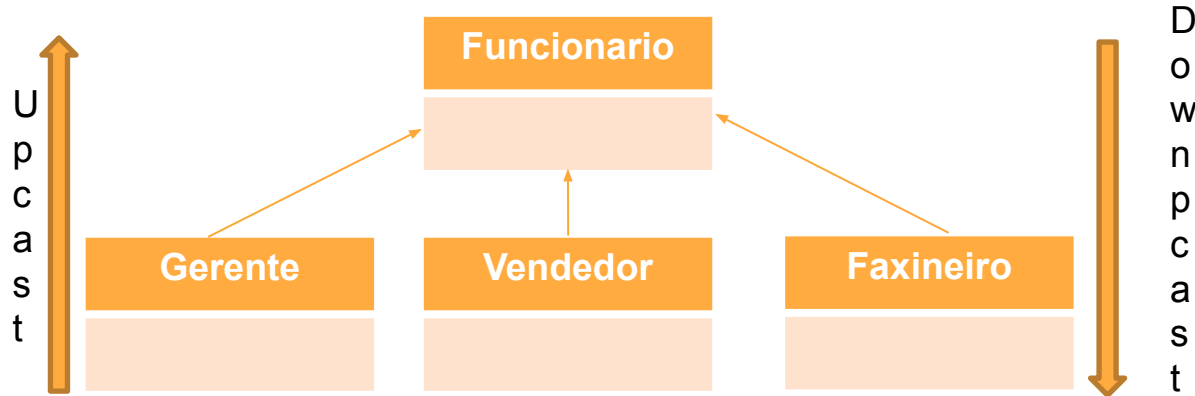
- C++

Python

```
class A (B, C) :  
    pass
```


Herança

- Upcast e Downcast:



Herança

- Upcast:

Java

```
A a = new B ();
```

C#

```
A a = new B ();
```

Python





Herança

- Downcast:

Java

```
B b = (B) new A();
```

C#

```
B b = (B) new A();
```

Python



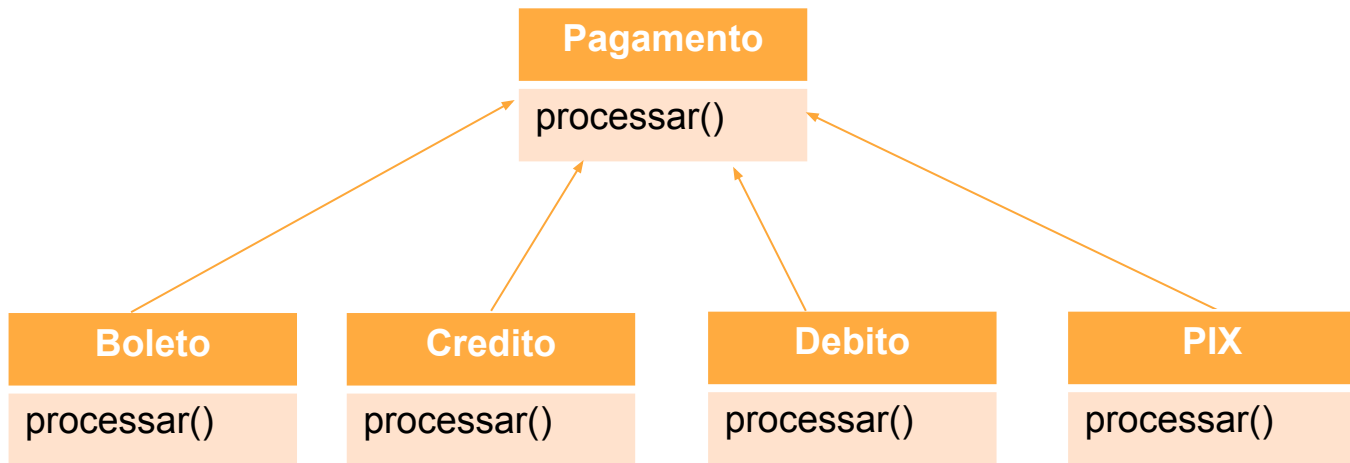
Herança

- Polimorfismo

"A mesma ação, se comportando diferente."

Herança

- Polimorfismo



Herança

- Polimorfismo
 - Códigos:
 - Ver exemplo

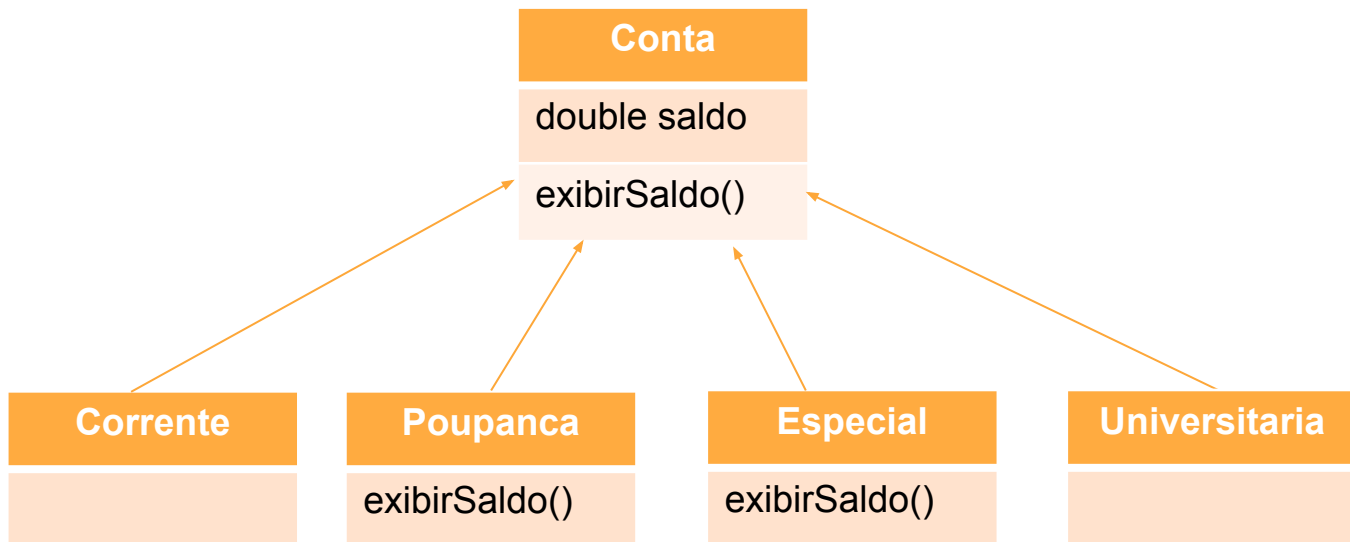
Herança

- Sobrescrita

"A mesma ação, podendo se comportar diferente. "

Herança

- Sobrescrita



Herança

- Sobrescrita
 - Códigos:
 - Ver exemplo

Curiosidade

Polimorfismo x Sobrescrita

Exercitando 2

Crie as classes "Funcionario", "Gerente", "Vendedor" e "Faxineiro".
Realize upcasts e downcasts.

Obs: use a linguagem que gostar e siga as dicas sobre como criar classes. Faça a relação de herança de acordo com o slide.

Exercitando 3

Analise do comportamento de Polimorfismo e Sobrescrita.

Associação

"Possibilita um relacionamento entre classes/objetos, no qual estes possam pedir ajuda a outras classes/objetos e representar de forma completa o conceito ao qual se destinam. Neste tipo de relacionamento, as classes e os objetos interagem entre si para atingir seus objetivos."

Associação

- Tipos:
 - Estrutural
 - Composição
 - Agregação
 - Comportamental
 - Dependência

Associação

- Tipos:
 - Estrutural
 - Composição: "Com Parte Todo"
 - Ex: Pessoa e Endereço



Associação

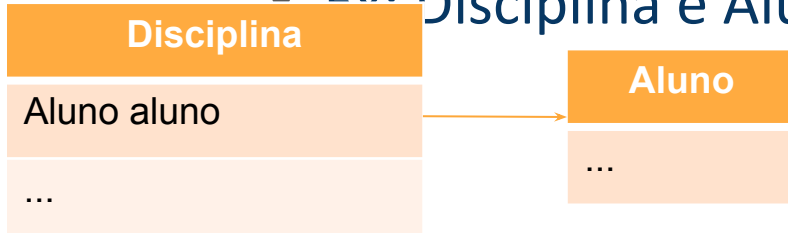
- Tipos:
 - Estrutural
 - Composição: "Com Parte Todo"
 - Ex: Pessoa e Endereço

Java

```
class Pessoa {  
    Endereco endereco;  
}
```


Associação

- Tipos:
 - Estrutural
 - Agregação: "Sem Parte Todo"
 - Ex: Disciplina e Aluno



Associação

- Tipos:
 - Estrutural
 - Composição: "Sem Parte Todo"
 - Ex: Pessoa e Endereço

Java

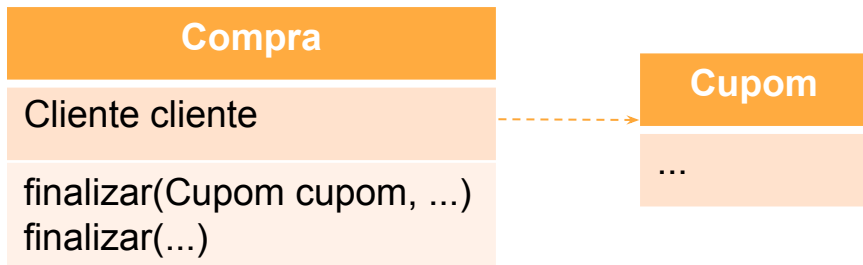
```
class Disciplina {  
    Aluno aluno;  
}
```

Curiosidade

Agregação x Composição

Associação

- Tipos:
 - Comportamental
 - Dependência: "Depende de"



Associação

- Tipos:
 - Comportamental
 - Dependência: "Depende de"
- Java

```
class Compra {  
    ...  
    finalizar(Cupom cupom, ...) {  
        ...  
    }  
}
```

Curiosidade

Herança x Associação

Exercitando 4

Apenas para praticar, codifique os exemplos dos slides anteriores sobre as associações.

Obs: use a linguagem que gostar e siga as dicas sobre como criar classes.

Interface

"Define um contrato que deve ser seguido pela classe que a implementa. Quando uma classe implementa uma interface, ela se compromete a realizar todos os comportamentos que a interface disponibiliza."

Interface

- Códigos:

Java

```
interface A {  
    ...  
}
```

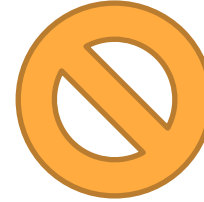
```
class B implements A {  
    ...  
}
```

C#

```
interface A {  
    ...  
}
```

```
class B : A {  
    ...  
}
```

Python



Exercitando 5

Apenas para praticar, crie uma interface chamada "OperacaoMatematica". Crie também 4 métodos das operações básicas: soma, subtração, multiplicação e divisão.

Obs: use a linguagem que gostar e siga as dicas sobre como criar classes. Tente não implementar algum dos métodos e veja o que acontece.

Siga em frente...

- Tipos de classe: Abstrata e Concreta
- Métodos abstratos
- Características das associações
- Palavras coringas: super, base e super()
- Relações entre classes e interface: extends e implements

Aula 5: A organização

Orientação a
Objetos

Objetivos

1. Apresentar os conceitos que organização a OO:
 - Pacotes
 - Visibilidades

Pacotes

"São uma organização física ou lógica criada para separar classes com responsabilidades distintas. Com isso, espera-se que a aplicação fique mais organizada e seja possível separar classes de finalidades e representatividades diferentes."

Pacotes

- Códigos:

Java

```
package ...;
```

```
import ...;
```

C#

```
namespace {...}
```

```
using ...;
```

Python

```
__init__.py(2.x)
```

```
from ...
```

```
import ...
```

Exercitando

Ver projeto de exemplo para entender funcionamento.

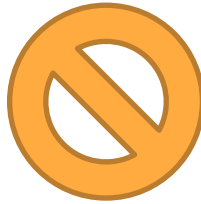
Visibilidades

"Um modificador de acesso tem como finalidade determinar até que ponto uma classe, atributo ou método pode ser usado. A utilização de modificadores de acesso é fundamental para o uso efetivo da Orientação a Objetos. Algumas boas práticas e conceitos só são atingidos com o uso corretos deles."

Visibilidades

- Tipos:
 - Private
 - Protected
 - Public

Python





Visibilidades

- Tipos:
 - Private: Só dentro a classe
- JavaC#

private

```
private int i;  
private void do();
```

private

```
private int i;  
private void Do();
```

Visibilidades

- Tipos:
 - Protected: Dentro da classe, mesmo pacote e subclasses
- Java

C#

protected

```
protected int i;  
protected void do();
```

protected

```
protected int i;  
protected void Do();
```

Visibilidades

- Tipos:
 - Public: Em qualquer lugar

Java

C#

public

```
public int i;  
public void do ();
```

public

```
public int i;  
public void Do ();
```

Curiosidade

Qua usar?!

Exercitando

Ver projeto de exemplo para entender funcionamento.

Aula 6: Próximos passos

Orientação a
Objetos



- Padrões de Projeto(Desing Patterns)
- Boa práticas: SOLID, código, tecnicas de programação, etc.
- Refatoração
- UML
- Frameworks
- MUITA prática e estudo!

Dúvidas?

- > Fórum do curso
- > Comunidade online (discord)

Para saber mais

- <https://www.casadocodigo.com.br/products/livro-oo-conceitos>